EFFICIENCY CONSIDERATIONS BETWEEN COMMON WEB APPLICATIONS USING THE SOAP PROTOCOL

Roger Egggen, Sanjay Ahuja, Paul Elliott
Computer and Information Sciences
University of North Florida
Jacksonville, FL 32224
USA
ree, sahuja, ellp001@unf.edu

Maurice Eggen
Department of Computer Science
Trinity University
San Antonio, TX 78212
USA
meggen@trinity.edu

ABSTRACT
Applications can communicate over the WEB using a variety to tools and techniques. Both Java and PERL are common languages used in WEB communication. A Simple Object Access Protocol (SOAP) is used within the PERL and Java languages to determine which is the most efficient communication technique.

KEYWORDS
PERL, Java, and Distributed Applications

1. Introduction
WEB communication can occur in a variety of ways using different techniques and tools. Distributed processing belongs to one of the most important areas of research in our modern computing world. In this paper, we discuss the use of PERL (Practical Extraction and Report Language) compared to Java both using SOAP (Simple Object Access Protocol). PERL is an easy to learn quick prototyping language and is used in many WEB applications. We then introduce Java as a tool that supports the same communication protocol between workers, that of SOAP. We compare the speed and efficiency of PERL and Java as they implement the SOAP protocol.

2. Fundamentals
A distributed system is a collection of independent processors interconnected by a network and capable of cooperating to achieve the solution of a task. The task could be widely varied including motion, object detection, recognition, or a wide variety of other tasks that cooperate to arrive at a desired conclusion. By distributing the tasks to a wide variety of processors, we reduce the amount of processing at a central site. The system is distributed, data is decentralized, and computation and communication among workers asynchronous. Each worker will process the data related to its current configuration and report the results to a the central processor. The processing workers are considered loosely coupled if they do not share memory and have individual processing power, hence the idea of parallel distributed processing across the WEB. Important strides have been made recently to provide environments that make distributed computing more common and easier to implement. The message passing interface (MPI), parallel virtual machine (PVM), Common Object Request Broker Architecture (CORBA), Java’s Remote Method Invocation (RMI), and others have extended ordinary languages to provide methods (functions, procedures) facilitating communication between distributed computing workers. PERL and Java are often used in distributed WEB applications. These languages are easy to program and have features that support distributed communication.

This paper studies the SOAP protocol for Java and PERL. We compare features and efficiency of both in distributed communication.

3. PERL
PERL is an open source, cross-platform programming language capable of performing a wide variety of applications. The word PERL is an acronym for “Practical Extraction and Report Language.” Used in both the public and private sectors, PERL is supported by UNIX, Macintosh, Windows and many more operating systems. PERL comprises the best features of many languages, including C, awk, sed, sh and BASIC. PERL works with
Application servers
- Artificial intelligence algorithms
- Astronomy
- Audio
- Bioinformatics
- Compression and encryption
- Content management systems (for both small and large scale Web sites)
- Database interfaces
- Date/Time Processing
- eCommerce
- Email processing
- GUI development
- Generic algorithms
- Graphing and charting
- Image processing
- Mathematical and statistical programming
- Natural language processing (in English, Chinese, Japanese, and Finnish, among others)
- Network programming
- Operating-system integration with Windows, Solaris, Linux, Mac OS, etc.
- PERL/Apache integration
- Spam identification
- Software testing
- Templating systems
- Prototyping for fast development
- Text processing
- Web services, Web clients, and Web servers
- XML/HTML processing [1]

The widely available free modules from CPAN (www.cpan.org) support PERL applications development. These modules make PERL a good choice for development of distributed systems.

4. Java

Java is easy to learn and an effective approach for meeting the requirements of distributed processing. The object oriented capabilities of Java make it a robust, efficient, and therefore desirable environment in which to program the parallel communication

The Java interpreter and the extensive standard library are freely available in source or binary form for all major platforms from java.sun.com. The same site also contains programs, tools, and additional documentation. Java is object oriented with a class structure that encourages reuse. Java supports single inheritance, polymorphism, and all the other features expected of an object oriented language.

The Java Virtual Machine is easily extended with new functions and data types implemented in either Java or by native routines that can be implemented in C or C++ (or other languages that can be invoked from C).

5. SOAP

The XML syntax of a SOAP message is uncomplicated. A SOAP message consists of an envelope containing: 1) an optional header containing zero or more header entries (sometimes ambiguously referred to as headers), 2) a body containing zero or more body entries, and 3) zero or more additional, non-standard elements.

The only body entry defined by SOAP is a “SOAP fault” which is used for reporting errors. Some of the XML elements of a SOAP message define namespaces, each in terms of a URL and a local name, and encoding styles; a standard one is defined by SOAP.

Header entries may be tagged with the following optional SOAP attributes: 1) an actor which specifies the intended recipient of the header entry in terms of a URL, and 2) an indication that specifies whether or not the intended recipient of the header entry is required to process the header entry.

6. Hardware

The hardware for evaluating SOAP efficiency consists of a Beowulf cluster of computers all running RedHat linux v9. The machines are 0.5 gHz machines with 512 megabytes of main memory connected by gigabit fast ethernet.

7. Software

The software consists of Red Hat Linux 9 core 3.2.2-4, PERL 5.8.3 installed with threading, and Java HotSpot 1.4.1_01. SOAP::Lite 1.2 for PERL was used in the PERL communication and JAXRPC, jwsdp-1_3_01, was used in the Java communication. Only Linux machines were involved to remove discrepancies caused by different operating systems.

8. Programming a Distributed System

Regardless of the language, programming a distributed system, involves one of several fundamentals. Typically, one of three scenarios are involved: boss/worker where the boss distributes a portion of the work uniquely,
worker crew where each processor does essentially the same processing on different portions of the data, often referred to as single instruction multiple data (SIMD), and a pipeline where each worker processes a portion of the data, passing that partial solution to the next worker. In this research we use the boss/worker environment as shown in Figure 1. A boss process defines the task to be accomplished and decides the method of distribution. Several worker processes do the actual work. The worker starts by receiving input from the central server (boss). The boss is the controlling process that has the responsibility of the overall system. A portion of the task is given to each of the workers. The boss waits to receive each of the completed tasks from the individual workers. Each of the workers execute in parallel.

![Figure 1](image-url)

**Partial Communication Dependency**

### 9. Evaluation

The purpose of this research is to determine how much a user gains (or gives up) to enjoy the ease of programming language PERL versus Java in a distributed environment using SOAP. We chose a simple sorting algorithm as the activity performed by each worker. To assess relative efficiency, we held the activity at each of the workers relatively constant. We programmed an n² sort so reasonable time would be taken on each of the worker processors, creating a measure of both communication and computation times. Boss and workers are programmed using Java and PERL using the SOAP application programming interface. The PERL side of the test used middleware called SOAP::Lite, while the Java side of the test used JAXRPC. Each application is implemented with exactly the same functionality. The boss machine creates sets of integers whose cardinality vary between 1,000 and 256,000. Each set was evenly divided and distributed among the worker machines. Each worker sorted their portion of the data, yielding a highly parallel solution. The communication from boss to worker to boss was handled through the SOAP protocol.

As is often the case, the boss did not participate in the primary task, but rather distributed the work and waited for results. Timings included establishing communication, dividing the data, distributing the data, sorting each data set, receiving the sorted results from each worker, and merging to arrive at a totally sorted set of data.

### 10. Results

We ran datasets of sizes 1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000, and 256000. Each were sorted and timed. The timing results are in Figure 2. The Figure shows a clear cross over between communication and processing dominating the timings. PERL processes much slower than Java, thus when significant amounts of data are present at each worker, PERL requires more time. But, PERL establishes communication faster than Java. Figure 3 more clearly shows the relative efficiency for 4 workers.

Some comments related to PERL are in order. In our preliminary tests, we found that the efficiency of PERL is significantly less than that of Java. A variety of considerations were employed to increase PERL’s efficiency, each with limited success. PERL 5.8.0 ran 90% as fast as PERL 5.8.3, so one should certainly ensure the latest version of PERL is used. Also, threading is relatively new to PERL, only coming into existence in the 5.8.x versions. Threaded PERL 5.8.3 runs 96% as fast as a non-threaded installation. In a distributed application, installation of a threaded version of PERL is essential. PERL 6 is soon to be released and hopefully, efficiency will be addressed. With these considerations in mind, we implemented the PERL application by forking new processes rather than using threads. PERL’s forking mechanism is mature and well developed, therefore, until PERL 6 is available which we believe will provide more stable threading, forking is the method of choice.

The PERL application can be compiled to native code using perlcc. Perlcc is in a “nonstable” state and should not be used for production applications, but it did generate native code. However, the native code executed only 92% as fast as the code directly interpreted. We expect a significant increase in performance when perlcc is enhanced.

A less defined measure is programmer productivity. We carefully monitored our development time for each of the applications. While this is somewhat arbitrary, ill defined, and rather qualitative, we believe our experience is suggestive of the nature one can expect when considering these environments. Upon evaluation of our respective experiences in each environment, we believe programmer time was most efficient in PERL and with development of the Java application approximately 30% longer. We realize this consideration is highly arbitrary, but believe programmer productivity is a significant factor in cost of application development.
<table>
<thead>
<tr>
<th>Workers</th>
<th>2</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Size</td>
<td>PERL</td>
<td>Java</td>
<td>PERL</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>13</td>
<td>4</td>
</tr>
<tr>
<td>2000</td>
<td>14</td>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>4000</td>
<td>50</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>8000</td>
<td>115</td>
<td>15</td>
<td>34</td>
</tr>
<tr>
<td>16000</td>
<td>347</td>
<td>21</td>
<td>108</td>
</tr>
<tr>
<td>32000</td>
<td>2980</td>
<td>49</td>
<td>894</td>
</tr>
<tr>
<td>64000</td>
<td>9257</td>
<td>167</td>
<td>5682</td>
</tr>
<tr>
<td>128000</td>
<td>17787</td>
<td>696</td>
<td>11612</td>
</tr>
<tr>
<td>256000</td>
<td>74705</td>
<td>2830</td>
<td>46448</td>
</tr>
</tbody>
</table>

**Figure 2**

Efficiency Timings

However, when developer productivity, ease of programming, portability, and prototyping are desired, PERL and Java are viable alternatives. PERL and Java support a robust and convenient API for implementing distributed, Web based, or distributed intelligent agent applications.

Java’s communication is efficient, which accounts for some of its performance. Also, interpreting byte code is more efficient than pure interpretation. Note, each application scales appropriately with the size of the data set.

We note that initially PERL is more efficient than Java in establishing communication. This accounts for the lower timings PERL received when small amounts of data is present. However, Java quickly out performs PERL when more computation is required. PERL is quite inefficient when any degree of processing is required. This leads to a decision for software developers. If establishing communication is the primary purpose of the application, PERL is a very good choice. If any significant amount of processing is required after communication is established, Java is the language of choice.

**11. Summary and Conclusions**

The above timings were taken during a period in which the network and processors were not used by others.

If the fastest application processing possible is the goal, PERL or Java would not be the choice. PVM or MPI with native routines written in C will outperform all the above applications.

**12. Bibliography/References**

[1] [http://www.perl.org/](http://www.perl.org/)

[2] [http://java.sun.org](http://java.sun.org)


